Why and how to investigate interaction design of software development tools

Juliana Jansen Ferreira[1,2], Clarisse Sieckenius de Souza[1] and Renato Cerqueira[1,2]
[1]Semiotic Engineering Research Group (SERG)
Departamento de Informática, PUC-Rio
Rio de Janeiro – RJ, Brasil
[2]IBM Research Brazil
Rio de Janeiro - RJ, Brasil
{jjansen, rcerq}@br.ibm.com, clarisse@inf.puc-rio.br

***Abstract*— The existence of some relationship between the usability of software development tools and the quality of end users' interaction with the product these tools contribute to build would not be surprising. Should this be the case, a developer's problematic use experience with these tools would increase the workload of HCI experts, whose aim is to promote high quality user experience with software products. Yet, this connection has not deserved much attention from researchers, and it is unclear how investigations should be conducted to verify if it is true. Our contribution in this paper is a first step in this direction. We propose an inspection method to characterize communicability and usability aspects of software modeling tools. By combining both aspects and articulating our analysis around tool, notations and people, we provide valuable conceptual links that, we argue, may in the long run of subsequent research contribute significantly to verify the (extent of the) relation between HCI quality of development tools and developed products.**

***Keywords*—Human-computer interaction; software development tools; notations; cognitive dimensions of notations; usability; communicability; semiotic engineering**

I. INTRODUCTION

Software development is heavily supported by tools. From the design phase through the testing phase, various tools are used to support people in producing intermediary artifacts as well as the finally achieved piece of software. The quality of development tools can certainly impact the quality of produced software in a number of ways, including the end user's interaction experience. What we do not know is the potential magnitude of this impact, where and how it originates, and finally how negative impacts might be avoided (probably by improving the quality of development tools themselves). The challenge for researchers who want to know the answers to these questions is that software development tools – and modeling tools in particular – are very complex, hard to design and hard to evaluate [1].

Our long-range research goal is to investigate specifically if some of the breakdowns that end-users experience while interacting with computer technologies are related to breakdowns that software designers have experienced themselves while using software development tools to produce such technologies. We frame this question as a conjecture, named the *Propagation Conjecture*. The very first step in the long way towards finding an answer to the question is to identify how, where, when and why interactive breakdowns may happen while software designers are acting as users of software development tools.

Several types of tools are used along the development process (*e. g.* coding tools, database definition tools, etc.). We began our investigation with modeling tools because modeling is an important activity at some point in all software development projects [2]. Methodologically, we chose to start by inspecting modeling tools' interfaces. One of the reasons for it is that inspection methods are lower-cost evaluation aids than user observation methods. The other reason is that inspection methods – and in particular theory-based inspection methods, as we propose to use – can *frame* evidence collected during inspections in such a way that the analyst can more easily establish certain relations among various instances and kinds of data. These relations naturally feed the elaboration of inferred abstractions from meaningful correspondence verified between types of evidence, which can be translated into qualified hypotheses for experimental research projects or into qualified possibilities to be explored in subsequent qualitative research projects as well as in professional practice *cases*. Our evaluation of modeling tool interfaces addresses two specific aspects of human-computer interaction: usability and communicability. Whereas usability is a notion strongly rooted in cognitive HCI perspectives [3][4][5][6], communicability originates from a semiotic approach [7].

The evaluation of modeling tools needs to consider some particular characteristics of software modeling tasks, which involves three tightly related factors [8]. First, most professional software models are built with *tool* support. Therefore, tools can clearly influence the final quality of models composed with them. Second, software models are

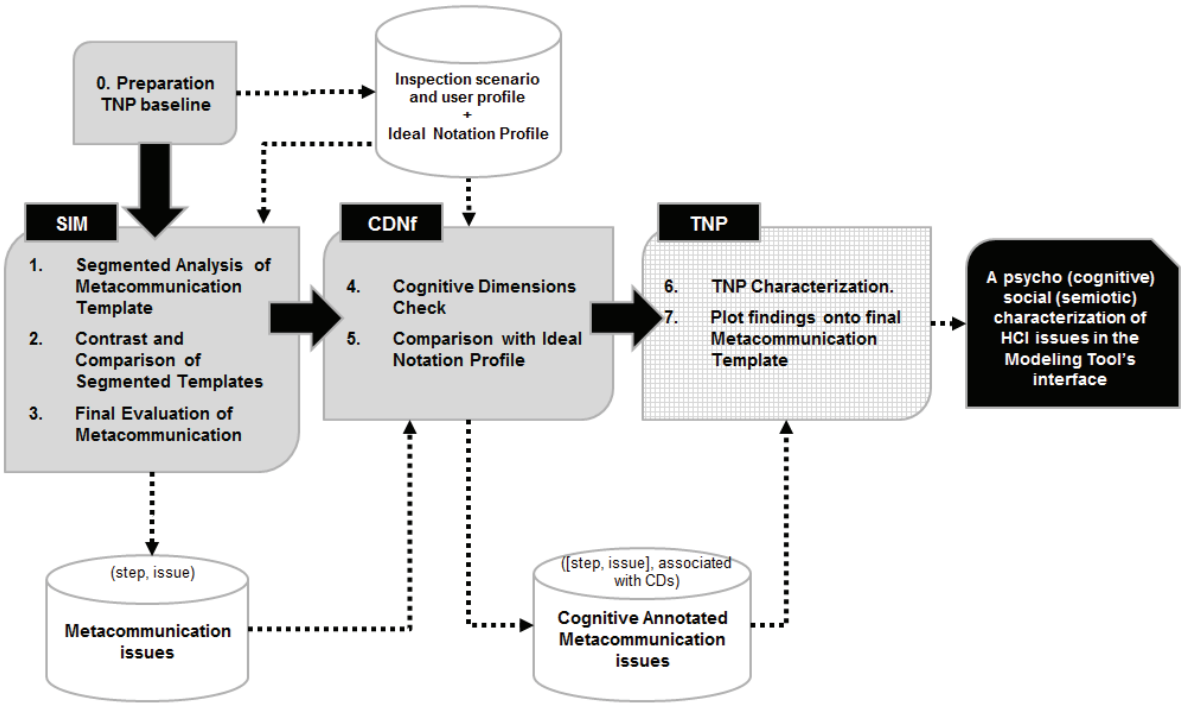

Fig. 1. Combined Semiotic-Cognitive Evaluation method (CSCE) using the Tool-Notation-People triplet (CSCE+TNP)

expressed by specific *notations*. OMG [9], for example, specifies the most widely used set of software development modeling notations, the Unified Modeling Language (UML) [10]. As a result, software modeling tools' designers need to work with notation specifications, including semantic definitions and current updates to provide proper support for software modeling tasks. Third, software models are built *by people* and usually also *for people*, who collaborate to build a single shared artifact: the final software product.

In this paper, we present the Combined Semiotic-Cognitive Evaluation method (CSCE). As implied, CSCE combines communicability and usability perspectives into an integrated inspection procedure. Its purpose is to evaluate user interfaces of modeling interface tools, that is, the product of an HCI design process aimed at attending to the needs of a special class of users – software modelers. The result is a rich multi-perspective analysis of human-computer interaction in this very specific but nevertheless critical context for a long-term investigation of the *Propagation Conjecture*.

Since our inspection must take into account special modeling characteristics previously mentioned, we use a structure with three interconnected dimensions to guide CSCE analysis, namely the Tool-Notation-People *triplet* (TNP). Moreover, we use the *Semiotic Inspection Method*, SIM [7][11][12] and the *Cognitive Dimensions of Notations Framework*, CDNf [13] as a basis for the communicative and cognitive analysis in CSCE. As a consequence, the complete inspection tool we propose is CSCE+TNP (Fig. 1).

Given our long-term research goal to verify the *Propagation Conjecture*, which is centered on breakdowns experienced by end users and software developers (see above), CSCE+TNP focuses on identifying and characterizing interaction *issues* in modeling tools. The results achieved with the proposed method give us a solid base from which to infer some of the potential consequences of developers' interaction with the tool in the broader context of software development.

We illustrate the power of CSCE+TNP with findings from an evaluation of Enterprise Architect Ultimate (EA) [14], a well-known and widely used software modeling tool. The illustration shows that, in addition to a seemingly richer and more contextualized interaction evaluation *per se*, the proposed method of analysis has the potential to provide HCI designers with an integrative conceptual framework with which to elaborate on important aspects of interactive design. Individually, none of the combined components can achieve comparable results.

The work reported in this paper builds on previous work published in a conference paper in 2014 [15]. The current version, however, presents considerable improvements compared to the previous one. Such improvements are the result of interim research. The CSCE version presented in this version is more robust and perspicuous than the previous one, mainly because we are now using SIM in its full-fledged form for the semiotic analysis, rather than just a fill-out of the designer-to-user message template. In the first author's Ph.D. thesis [16], the interested reader will find inspections of four software modeling tools using the combined method as presented in this paper.

The next sections are organized as follows. We begin by briefly discussing the broader investigation context where we

are positioned. Then we describe the individual HCI perspectives with which we explore usability and communicability features in this particular domain. Next, we describe our proposed method in detail and illustrate the main results we have achieved with an evaluation of EA. In the last two sections, we discuss some of the implications of our current findings and outline future work on the path towards investigating the *Propagation Conjecture*.

II. Why investigate Software Modeling Tools

The *Propagation Conjecture*, as defined in the opening paragraphs of the introduction above, may seem trivial and uninteresting at first sight. Some may find it obviously true that if software developers have to deal with poorly designed software engineering support tools, this may eventually influence the quality of the final software product in one way or another. Moreover, probably because most software developers will admit that they have to deal, more or less frequently, with poorly designed interfaces, it may be tacitly assumed that part of the HCI experts' job in formative or summative evaluations of the end user interface is to absorb and eliminate problems that may have been caused by bad user experience with software development tools. Yet, such guesses about whether the *Propagation Conjecture* is true or false, and why, are in fact a weak argument against carrying out an extensive investigation, given the possibility that the impact of the propagation can be costly in terms of wasted HCI expert resources. For example, what if we find out that developers face numerous interaction problems with their tools and that, not only do such problems propagate and impact the end user's experience, but that they can actually be avoided if more effort is devoted to designing better interfaces for software development tools? This would be a grim scenario of wasted efforts. And the only way to know if we are justified in paying little attention to the *Propagation Conjecture* (and protected against grim scenarios such as this one) is to make the first step and investigate the quality of HCI design in software development tools.

Another reason to investigate the quality of HCI design in software development tools as a whole is the same as why we investigate it in computer technology in general: people do want and deserve to have better use experiences, especially if – as is the case with development tools – their productivity may be at stake because of human-computer interaction issues.

Among the various kinds of software development tools to be investigated, we begin with those that support modeling tasks. Modeling tools share some characteristics with other development tools that create notational artifacts, such as specification and programming tools. Compare to other tools, however, software modeling tools tend to privilege conceptual, meaning-intensive, design tasks rather than implementational, form-intensive, engineering tasks. Moreover, given that professional software development is typically a social process, in addition to their role in triggering a modeler's reflection about the characteristics of a particular system, models are also a means to communicate team members' ideas and decisions to one another. Since communication is one of the prime mechanisms to propagate meanings (and misunderstandings, too), software modeling contexts constitute a good starting point to investigate the *Propagation Conjecture*.

III. Usability driven and Communicability driven Inspections of Systems interfaces

The theoretical underpinning of our research comes from two HCI perspectives that have comparable depth, but significant contrasts: Semiotic Engineering [17] and Cognitive Engineering [5].

Semiotic Engineering, the theoretical foundation of our communicability-driven analysis, focuses on communication and signification processes taking place in human-computer interaction. Following the same line as the work of pioneers in semiotic approaches to HCI such as Nadin [18][19] and Andersen [20], Semiotic Engineering views HCI as a special case of computer-mediated human communication, in this case between designers (or systems conceptual *producers*) and users (or systems *consumers*). Thus, this theoretical framing brings HCI designers onto the stage of human-computer interaction, something that no other kind of HCI theory has done to-date. Signs are the main concept in Semiotic Engineering, which adopts Peirce's definition for them. In Peircean Semiotics signs are defined as anything that stands for something else, to somebody, in some respect or capacity [21]. Therefore, in Semiotic Engineering, systems' interface signs convey the systems' creators' message to the users [17]. Through their interfaces, systems thus behave as their designers' *proxy* during interaction, thus achieving mediated designer-user communication. The designers' communication to users is about how, when, where and why to communicate with the system to achieve various kinds of goals and effects. In other words, it is communication *about* communication or *metacommunication*.

Cognitive Engineering and Norman's Seven-Step Theory of Action [5], the theoretical foundation of our usability-driven analysis and one of the more widely known characterizations of user-centered human-computer interaction, focuses on users' physical and mental activity during interaction. According to this theory, systems project an *image* (representations and behaviors) that users can control and perceive by performing physical actions that are causally related to mental actions that users perform based on their intentions, plans, interpretations, and decisions. Human-computer interaction is characterized as an iterated cycle of seven steps: (1) the establishment of an overall goal; (2) the definition of an immediate intention to be achieved towards the overall goal; (3) the elaboration of a plan of actions; (4) the execution of such actions; (5) the perception of the system's reaction; (6) the interpretation of perceived reaction; and (7) an assessment of whether the overall goal has been achieved (if not, a new immediate intention is defined). In the Cognitive Engineering model, after the overall goal is established, the process of human-computer interaction is defined as the user's iterated *traversal* of two gulfs between user and system: the Execution Gulf (steps 2, 3 and 4); and the Evaluation Gulf (steps 5, 6 and 7).

The most relevant difference between the two HCI perspectives that we adopt is who are the people each one of them takes into account while characterizing the process of

human-computer interaction. While the only *agent* involved in Noman's 7 Step Theory of Action, which serves as the foundation for Cognitive Engineering, is the *user* (hence the *user-centered* perspective it defines with great clarity), in Semiotic Engineering there are actually three *agents* involved in interaction: systems builders (especially HCI designers); the users; and also the systems, themselves, which represent their builders at interaction time. Therefore, in Semiotic Engineering, unlike in Cognitive Engineering, systems designers and systems user are equally important parts to be accounted for in the study of our phenomenon of interest. Moreover, although systems are not *human* agents, they play the same prominent role as systems designers and systems users, in Semiotic Engineering, since it is *through their mediation* that the other two achieve communication.

The combination of communicability-driven and usability-driven perspectives promotes a more comprehensive inspection of relevant parts of the object of analysis and the relations between them. However, as already mentioned, CSCE+TNP inspections focus on identifying and characterizing interaction *problems*, which is a special way of using the combined methods, namely the Semiotic Inspection Method (SIM [7] [11][12]) and the Cognitive Dimensions of Notation framework (CDNf [13]). Both methods, presented in further detail below, have been originally designed to support an assessment of HCI problems and merits alike.

A. Inspecting Communicability with SIM

Every communication process involves senders and receivers. SIM is a five-step inspection method (Fig. 2) where we look at the *sender's* activity[1]. As a top-level description, we can say that the method deconstructs the designers' message, analyzes deconstructed parts, and finally reconstructs the message. In the deconstruction and reconstruction processes, the designers' message and communicative strategies become visible to the analyst, who can then appreciate the quality of metacommunication. The deconstruction phase is achieved with a segmented analysis of the interaction designers' message. SIM analyzes the *emission* of the designers' metacommunication, that is, it looks at the message itself and produces a rich description of what it is *telling* to its receivers, and how.

In the *segmented* analysis, metacommunication is analyzed in layers, each one of them geared by one of the three sign classes proposed by Semiotic Engineering: Metalinguistic, Static and Dynamic signs. ***Metalinguistic signs*** are interface signs that refer to other interface signs and provide information, explanation, illustration or warnings about these. A typical example of metalinguistic signs is what users see in screen tips, usually a short phrase (verbal signs) communicating what a button, an icon or an information field *means*. ***Static signs*** are non-metalinguistic interface signs whose meaning users can productively interpret without having to engage in interaction. They are the ones that can be seen in interface *snapshots*, which instantly communicate something to users. Finally, ***dynamic signs*** are those found in a temporally or causally related sequence of interface signs. Unlike static signs, which users can fully interpret as soon as they see them, dynamic signs only make sense to users *over time*, that is, as users engage in interaction and see what happens (*i. e.* see the designers' message unfold through interaction). The time it takes for users to make sense of dynamic signs can vary. Sometimes a simple before-after sequence will be enough to tell what a dynamic sign means. Other times the user will have to go as far as to open a dialog or even effect some action (and then possibly 'undo' it) in order to understand what the designers are trying to say. This is the essence of dynamic signs: they only communicate what they mean through interaction. Therefore 'the sign' is the entire sequence, made of various static signs and possibly also by metalinguistic signs.

The above classification of signs, as is the case of most classifications in interpretive methods like SIM, is the result of the inspector's judgment. For example, there is no automatic way to determine *a priori* whether a button with the label 'Cancel' is a static sign or part of an unfolding dynamic sign when users of an e-commerce application find it during a purchasing process. For instance, if they 'Cancel' the third step in the process, does it mean that they go back to the second step? Or does it mean that all previous steps will be canceled? If metacommunication (especially the one achieved with metalinguistic signs, in this case) is not clear, the user will have to 'learn by doing'. In other words, the *meaning* of the static label 'Cancel' may not be enough to communicate the designers' entire message. Hence, the inspector should be alert to whether meanings anticipated by static signs are confirmed (or not) by the associated dynamic signs.

The comment in the previous paragraph also points at a critically important requirement in SIM: that the inspector has clearly defined the profile of the user to whom the designers are sending their message (*i. e.* the intended receiver of the message), as well as a purpose for the entire communication (*i. e.* the task or activity that metacommunication should achieve) as well as the context in which it takes place. Unless these factors are clearly established, decisions about what interaction signs mean (to intended users) are vacuous. This requirement is common to many other HCI inspection methods and is attended to at the preparation step shown in Fig. 2.

[1] The Semiotic Engineering method focused on *receivers* is CEM, the Communicability Evaluation Method [7].

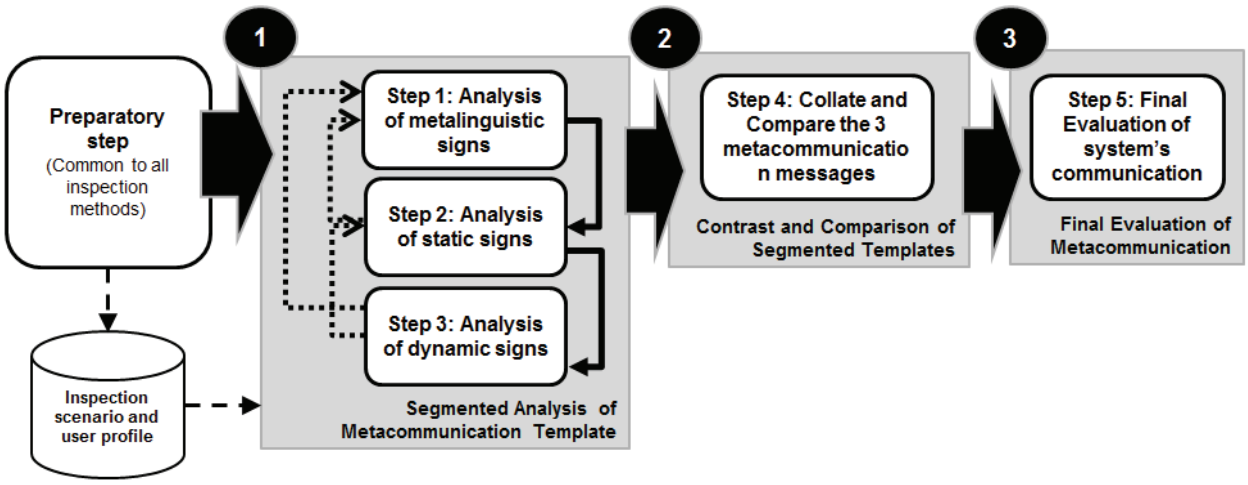

Fig. 2. Semiotic Inspection Method - SIM

In order to meet inspection requirements, in the ***preparatory step*** (Fig. 2) the inspector carries out an informal walkthrough of the selected artifact, aiming at establishing the focus of analysis. The evaluator determines the elements and conditions of communication processes, in general, namely: the senders, receivers, message contents, message codes, communication channels and context. Since communication is meant to achieve the users' goals, evaluators must clearly identify the targeted users of the system and the top-level goals and activities that the system supports. The result of the preparatory step is an inspection scenario, in which the designers' metacommunication is being sent to an identified receiver (a targeted user with a clearly established profile), in the context of a clearly established activity, for a clearly established purpose.

The semiotic inspection proceeds with phase 1, the three-step ***Segmented Analysis of Metacommunication Template***, an abstract representation of the designers' message to the users. The content and structure of the metacommunication template, which the analyst iteratively composes for every class of sign, is the following:

"Here is my understanding of who you are, what I've learned you want or need to do, in which preferred ways, and why. This is the system that I have, therefore, designed for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision." [17].

The first person of discourse in the template (referred to as "I", "my") stands for the designer and the second (referred to as "you", "your") stands for the user. The third person in discourse (referred to as "it", "this") stands for the system, represented by its interface. Together, they characterize the participants in an elaborate metacommunication process that takes place during human-computer interaction.

At each one of the three steps in this phase, the analyst inspects metacommunication focusing on a single class of sign (metalinguistic, static and dynamic), hence the 'segmentation'. In step 1, the inspector takes **metalinguistic signs,** which has already mentioned refer to other interface signs, static, dynamic, or even metalinguistic. With metalinguistic signs, designers explicitly communicate to users the meanings encoded in the system and how they can be used. Thus, in step 1 the inspector will determine which parts of the metacommunication template can be filled out exclusively with meanings expressed by metalinguistic signs. Note that there may be gaps in the template, that is, parts of it that the designers have chosen *not to communicate* in the form of help information, explanations, illustrations, screen tips, warnings, and the like. This does not mean that there is a problem in metacommunication. It simply reveals the designers' choice.

In step 2 the inspector does the same thing with **static signs**. As already mentioned, these are *non-metalinguistic* signs whose meaning can be interpreted independently of temporal and causal relations. In other words, the context of interpretation is limited to the elements that are present on the interface at a single moment in time. Some examples of static signs are layout structure, menu options and toolbar buttons. It is important to bear in mind that signs are classified as **static** only if they do not communicate explanation, information, tips, warnings and illustration referring to other signs. If they do, they must be classified as **metalinguistic** signs. At the end of this step, like in the previous one, the inspector fills out the metacommunication template with messages expressed through static signs. Again there may be gaps in the template.

Finally, in step 3 the inspector iterates the process now looking only at **dynamic signs**, that is, signs that require interaction, and hence time, to make sense to users. They emerge with interaction and must be interpreted with reference to it. Strictly speaking, all static signs in an interface are part of some larger dynamic sign. For example, when a user selects the option "save as . . ." of a menu "file," systems typically exhibit a dialog window with a conversation about the file's name, location, format, etc. The causal association between the menu selection and the dialog that follows it is a dynamic sign. However, the communication conveyed by the static signs at the beginning of the interactive sequence is usually equivalent

to the one that unfolds in a subsequent interaction. In other words, metacommunication receivers can easily infer the dynamic sign that will follow if they activate a static sign. Hence, for a semiotic inspection, meanings that can only be expressed over time ([7], p. 19) are the most relevant dynamic signs to take into consideration. They *tell* the inspector which elements of the metacommunication template will only be communicated if users actually engage in interaction, that is, which part of the designers' message cannot be inferred from the system's single states alone. Note that once the user has encountered the first instance of a particular message communicated through a dynamic sign, he may well infer its occurrence the next time he sees the static sign that triggers it. This inferential process is the semiotic account of the user's *learning* of an interface language, which means that in time, as users become familiar with an interface, they will have to rely less on dynamic signs and metalinguistic signs. Static signs will become more and more meaningful to them. Not also that, for this reason, the inspector must be absolutely clear about the metacommunication receiver's profile in his analysis. An experienced user will interpret an interface in a different manner than a novice does.

Once the entire inspection scenario has been analyzed focused on **dynamic** signs, the inspector again fills out the metacommunication template with content communicated only through dynamic signs, as defined above. So, by the end of phase 1 the analyst will have filled out three independent metacommunication templates. All of them may have gaps, which has already mentioned do not necessarily mean that there is a problem in metacommunication.

In phase 2, ***Contrast and Comparison of Segmented Templates,*** the inspector takes the three metacommunication templates from the segmented analysis phase and looks at crucially important aspects of what metacommunication is saying and how it is saying it. First, it will be possible to check if all three segmented messages are consistent with each other. Consistency is not the same as identity or equivalence. Therefore, we are not trying to see that the messages *are the same*, but only that they do not contradict each other. Second, it will be possible to check which messages figure in more than one segmented instance of the template, which is an indication of redundancy, a powerful communication strategy. By the same token, it will be possible to see which ones figure in only one of the three templates, that is, which ones (if any) are communicated by a single class of signs. The inspector will thus be able to appreciate how the designer distributes communication of complex messages among the three classes of signs. For example, it will be possible to see which parts of the message are communicated with (or without) the use of supporting metalinguistic signs, which parts are mainly achieved with the use of dynamic signs, and so on. All of these have a direct impact on the message receiver's (the end user's) experience.

With the result of contrasts and comparisons, in phase 3, the ***Final Evaluation of Metacommunication***, the inspector can *reconstruct* (*i. e.* integrate the three templates into one) the designers' metacommunication message, paying attention to consistency, redundancy, and distribution, which should all make sense in view of the targeted receivers. The inspector can also identify which metacommunication strategies have been used and decide how well they are likely to achieve the designers' intent when addressing the targeted user population. In this last phase it is important to bear in mind that the concept of *targeted users* in Semiotic Engineering are the equivalent of the *ideal receivers* (or *readers*) in semiotic traditions that are interested in human communication. Although messages (in our case interface messages) exchanged by interlocutors are open to indefinitely many interpretations, "interpretive operations […] are by no means indefinite and must be recognized as imposed by the semiotic strategies displayed by the text", that is, the way the message is expressed ([22], p. 36). Thus, SIM application may occasionally reveal that the *reconstructed* targeted users – that is, the ideal receivers that are inferred from the message itself – do not coincide with the *intended* targeted users. This is the most severe type of communicability problem for Semiotic Engineering, which SIM can contribute to solve by detecting the origins of the problem and the logical chain of signs that must be changed in order to communicate a different message.

Compared to other inspection methods commonly used in HCI [23], SIM yields a precise characterization of who are *in fact* the designers' interlocutors in the process of metacommunication (first part of the template). It also acknowledges explicitly the possibility of creative user appropriations enabled by the designers' vision (last part of the template). Most HCI methods concentrate on how the system works and how it is (or can, or must be) used.

B. Usability-driven perspective inspection

CDNf is a prime candidate for evaluating the usability of tools dealing with notations. It defines a set of design principles for creating or evaluating notations, user interfaces and programming languages used with information artifacts. This framework provides a common vocabulary for discussing many cognitive factors of such representation-building systems. Its aim is to improve the quality of discussions and decisions in design and evaluation activity [13]. CDNf has been used to analyze notations in a variety of contexts, all of them related to systems where notations play a central role [13][24][25].

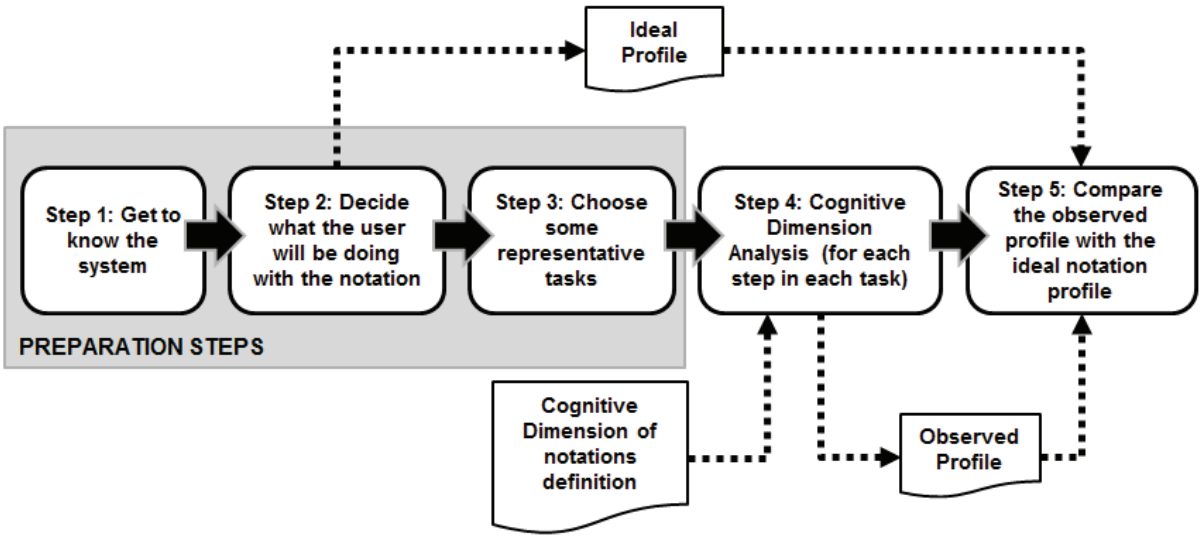

Fig. 3. Cognitive Dimension of Notation framework

The evaluation of a notation system design with CDNf is carried out in five steps, as shown in Fig. 3. In ***step one,*** the analyst gets to know the system and all of its components and all *information artifacts*, which are composed by one or more notations. In ***step two***, the analyst decides what the user will be doing with identified notations. In the ***step three,*** he chooses some representative tasks to serve as a reference for the inspection itself, which amounts to a verification of how notations behave or qualify in terms of fourteen *cognitive dimensions* [13] presented in TABLE I.

Tasks are typically achieved by means of subtasks. The first three steps of CDNf are the usual **preparation steps** of most inspection methods. Thus, in the ***step four***, for every subtask of every representative task, the analyst must ask whether the user can choose where to start, how an occasional mistake will be corrected, what will happen if there are second thoughts, what abstractions are being used, and so on. These questions spring directly from the definition of the fourteen cognitive dimensions (CD) used in CDNf. Moreover, once the user profile is defined, the CD can be taken as guidelines, like, for example: "Make it easy to change things"; "Make entities visible"; "Avoid constraining the order of doing things"; "Show relevant relation between entities"; and so on. The clear definition of a user profile (and the tasks to be achieved) is important because guidelines can change depending on who the user is and what he or she is trying to do. For instance, there may be reasons to wish for *viscosity*, that is, for "making it difficult to change things". Think of games, for example, where the challenge for advanced gamers would be to make it *difficult* to achieve changes.

For other examples that are closer to our domain of interest, we can take *Secondary Notation*. It refers to the situation when the inspected notation allows for the inclusion of additional information, typically using a different and often informal code. This is a way of expanding the notation. Therefore, during the inspection, the analyst will judge whether this expansion is possible, necessary, easy to express, easy to understand, and so on. Another dimension is *Error-Proneness*, which refers to the situation when the notation leads to mistakes, and the system (where the notation is used) gives little protection against them. During the inspection, the analyst will, therefore, seek to verify that notations *are not* prone to error.

TABLE I - LIST OF COGNITIVE DIMENSION OF NOTATIONS (ADAPTED FROM P.116-118, [13])

Cognitive Dimension	Description
Viscosity	Resistance to change
Visibility	Ability to view entities easily
Premature Commitment	Constraints on the order of doing things
Hidden Dependencies	Relevant relations between entities are not visible
Role-Expressiveness	The purpose of an entity is readily inferred
Error-Proneness	The notation invites mistakes and the system gives little protection
Abstraction	Types and availability of abstraction mechanisms
Secondary Notation	Extra information in means other than formal syntax
Closeness of Mapping	Closeness of representation to domain
Consistency	Similar semantics are expressed in similar syntactic forms
Diffuseness	Verbosity of language
Hard Mental Operations	High demand on cognitive resources
Provisionality	Degree of commitment to actions or marks
Progressive Evaluation	Work-to-date can be checked at any time

The iterated analysis in the fourth step, for all fourteen CD, will incrementally generate the involved notations' profile. Therefore, in ***step five***, the final one, the analyst compares the resulting observed profile with an ideal notation profile for the selected type(s) of activity defined for the inspection. Again, as

with SIM, an *ideal* standard is assumed by the inspector. Cognitive ideals can be provided by psychological theories that define the amount of mental effort required to achieve operations with notations. The smaller the mental effort, the higher the usability of the notation.

Compared with SIM, the use of CDNf is more straight forward. Different factors contribute to this. First, as is the case with most usability methods and the user-centered approach in general, the inspector is focused only on the user (not having to deal with designers and users, as well as with the relations between them that are established by metacommunication). Second, whereas SIM involves a deconstruction and a reconstruction phase in addition to the main analytical activity (*i. e.* the ***Contrast and Comparison of Segmented Templates***), once the preparation step is completed; the inspector is ready to carry out CDNf analysis. Third, whereas with SIM the inspector must explicitly classify all signs and explicitly determine what they mean, in CDNf the inspector can look at notations as a unit, an information artifact, not having to dissect how the compound is structured and how the various parts contribute to express what they mean. Having done this, his job is to determine how the artifact behaves or qualifies in terms of the fourteen CD, given the ideal behavior and qualification that targeted users should expect.

IV. The Combined Semiotic-Cognitive Evaluation Method and the TNP triple

In this section, we explain how we combined both methods and illustrated it step by step to show the kinds of results that we can achieve in evaluations guided by CSCE+TNP. The CSCE method, without the TNP triplet, has been shown to enrich HCI evaluation of different kinds of systems [26][27][28]. However, as already mentioned, the long-range goal of our research in the context of this specific paper is to be able to verify the conjecture that some of the problems that end users experience while interacting with computer technologies are related to problems that systems designers experience themselves while using software development tools. To this end, we have been focusing on the usability and communicability of modeling tools [15][16][27].

As briefly outlined in the introduction, the main features of the method are: (a) that it incorporates the core steps of existing semiotic (SIM) and cognitive (CDNf) inspection methods; (b) that the semiotic analysis is carried out before the cognitive analysis; (c) that given the long-range goal of our research, the analysis is centered on interactive problems detected in the process; (d) that an additional structuring resource (TNP) has been added to articulate how tools, notations, and people function are related to each other in the context of computer-supported modeling tasks; and finally (e) that the result of the combined inspection is a rich characterization of interaction issues in view of psychological (cognitive) and social (semiotic) dimensions of metacommunication carried out by modeling tools' interfaces.

Before we describe the process of analysis with CSCE+TNP, let us explain the roles of the semiotic and the cognitive analysis, as well as the role of the tool-notation-people triplet. The role of the initial semiotic analysis is to provide a broad contextualization of modeling tools. The metacommunication template [17], already presented in section III, is the main instrument for the combined analysis.

In TABLE II, we show the template subdivided into five portions: the designers' vision of who the users are **(A)**; what they believe the users need or want to do **(B)**; where, when, how and why **(C)**; as well as the correspondence between this vision **(E)** and the way how the artifact looks and behaves **(D)**. The metacommunication template, therefore, sets the keynote for the entire analysis.

TABLE II - Portions of the metacommunication template

A. Here is my understanding of who you are…
B. … what I've learned you want or need to do, …
C. … in which preferred ways, and why.
D. … This is the system that I have, therefore, designed for you, and this is the way you can or should use it …
E. … in order to fulfill a range of purposes that fall within this vision.

The role of the CDNf analysis, which focuses only on information artifacts directly associated with the issues found during the core steps of SIM analysis, is to expand the semiotic account of these issues with a cognitive account specifically centered on mental workloads required to work with the notations involved in communicative issues. Finally, the role of TNP is to structure semiotic and cognitive findings around the central elements of our HCI analysis (tool, notation and people) as applied to modeling tools' interfaces. In practice it serves as a classification resource, helping the analyst to organize findings in terms of which elements of the triplet, and relations between them, are affected by which ones of the semiotic and cognitive aspects of detected interface issues.

SIM evaluation deals with all elements of the triplet, but because of its communication perspective its strengths are more focused on "P", whose instances (designers, users and – depending on the designers' vision – other stakeholders and participants that affect, or are affected by, the users' activities) are brought together by "T". Although the designers' metacommunication message is encoded in signs (roughly equivalent to notations), unlike CDNf, SIM does not focus on individual units of information artifacts (each one having a specific notation system associated with it) [24]. Therefore, SIM provides a comparatively shallower characterization of "N".

Likewise, CDNf also deals with all elements of the triplet, although, as anticipated, in a completely different perspective. The "N" part is the main focus of a detailed analysis where multiple notational systems can be identified and decomposed into notational units. The "P" in CDNf, unlike in SIM, typically refers to a single person, the user. Finally, the "T" part in CDNf is virtually absent from the scope of analysis as

such. This is because tools are viewed as notational systems or information artifacts in CDNf, that is, "T" translates into "N".

The characterization of issues considering T, N and P presents a more broadly contextualized and detailed feedback to designers. It allows them to think about how local issues, with a particular interface item or notation feature, affect the users' overall activity and context. Hence the quality of new solutions – as well as the knowledge derived from it – can be better.

In the CSCE+TNP presentation that follows, we illustrate every step with significant findings and conclusions from a study carried out with Enterprise Architect (EA) [14], one of Gartner's Magic Quadrant of Enterprise Architecture tools [29]. The aim of the study was to investigate the various ways in which software modeling tools support software modelers. The results of the study, presented in complete detailed form in [16], were organized into four categories of breakdown issues:

1) **Problems with resources to extend/add functionality**.

2) **Problems with resources for meaning construction based on multiple notations**.

3) **Problems with resources for meaning construction based on UML alone**.

4) **Problems with supporting the software development process.**

As its name suggests, category 1 refers to semiotic and cognitive issues that spring from extension mechanisms provided by the most popular modeling tools among professional developers. Categories 2 and 3 could be organized as sub-categories of a single more abstract category (Problems with resources for meaning construction): in one of them meaning construction is supported by multiple notations; in the other by UML alone. Another possibility would be to keep only the more abstract category, dealing with all notations – UML and others – as part of the same problem. However, we decided to keep the distinction because of the general interest in UML, the most widely used modeling notation. The option to avoid subcategories was mainly due to the fact that this would be the only instance where subcategories would be used in the results of the EA study. Note that *meaning construction* refers to sense-making activities carried out both when creating a model and when interpreting a model (potentially created by someone else). Category 4 refers to the role played by models – the end product of modeling activity – in the broader context of software development.

For lack of space in the format of a journal paper, we selected evidence from category 1, ***"Problems with resources to extend/add functionalities"***, to illustrate all steps of CSCE+TNP, depicted in Fig. 1 (in the Introduction section). In the following subsections we present, describe and illustrate each step of the method, providing enough information for readers who are interested in applying CSCE+TNP in evaluation projects of their own.

A. Step 0 – Preparation – TNP Baseline

We start by defining the study's inspection scenario, which projects the research question ("in which ways do software modeling tools support software modelers") upon the territory of possible interactions with the chosen artifact(s) [30]. We must then select the tool to be inspected, the task(s) to be performed, and the targeted user profile(s). With these, we can define the TNP triplet baseline of the study. Note that because of the semiotic perspective of our method, the "P" portion in the baseline triplet must not only take into consideration the user(s) and possibly development team members with whom the users work or interact through the modeling tool itself or the product it creates, but also include the designers of the selected tool (who communicate with tool users through its interface). Moreover, the inspector must describe which relations are to be considered, in the specified scenario, among tool, notation, and people. These definitions serve as a reference for the entire inspection procedure. Note that, as shown in Fig. 1, this preparation provides the necessary inspection context for both, the semiotic (SIM) and the cognitive (CDNf) analyzes that follow. This is why the first three steps in the original CDNf method (all of them being *preparation* steps) can be skipped when transitioning between SIM and CDNf in CSCE. In the study with EA, the following definitions were established at the end of *Step 0*.

Inspection scenario. The chosen task was to build a UML activity model for a vacation request process, which is usually part of human resources management systems in large companies. The activity model should describe a flow that features a vacation request made by an employee, her manager's approval, the system notification and other subsidiary activities. The activity model is a UML behavior diagram that shows the flow of control or object flow, emphasizing the sequence and conditions of the flow. Activity models deal with both computational and organizational processes. Their role is that of an "interface model" between business and technology facets of the system being developed. Activity models can be used to describe either more abstract activities (like interactions between system and users or between systems) or specific object parts of the system, showing how such parts change along the process and over time [10]. This interface role of activity models was the reason for selecting them for our study scenario. They naturally span over a larger context of development decisions than other models.

User profile. The targeted user profile was that of an active professional in business modeling, requirements modeling, and conceptual modeling. However, we assumed that the user had never used EA (novice user for EA) and that he had little experience with UML *activity models* in real projects. All he knew about them was what he had learned in professional training and education, in the past.

TNP baseline relations. Our baseline triplet features EA as "T" and as a *proxy* for its designer (P_d). It also features the UML activity model notation as "N" and an active professional modeler (P_u), who doesn't have experience with EA (T) and knows of UML activity models (N) only in theory. The model consumer (P_{mc}) is also part of the baseline as "P". When P_u builds the model, he has an intended consumer for that model (P_{mc}). While producing the model, P_u takes P_{mc} needs into account, considering how the model will be used later as an artifact in the software development process.

B. *Step 1 – SIM Inspection: Segmented Analysis*

SIM starts with a segmented analysis of the designer's metacommunication message. The analysis is guided by Semiotic Engineering's three classes of signs: metalinguistic, static and dynamic signs. The analyst iteratively deconstructs the designer's complete message carried out by the interface into three segments: the metacommunication template message conveyed by metalinguistic signs alone; that conveyed by static signs alone; and finally that conveyed by dynamic signs alone. The results are three segments, none of them necessarily complete in itself. That is, segmented templates may have gaps of information. We filled out the metacommunication template for each class of sign, considering the portions indicated in TABLE II.

Metalinguistic signs analysis. The EA designers' metacommunication about extensions and add-ins is extensively conveyed through metalinguistic signs located in EA's website or in the user's guide that is installed with EA in the user's machine. In Fig. 5 we see that the designer is telling the user that extensions are developed by other people (Fig. 5-1). He also provides a list of recommended extensions (Fig. 5-2). However, he makes it clear that such extensions are something *other than EA* by referring the user to the third-party's website when it comes to explaining the details of the extension (Fig. 5-3). Add-ins are explained in the user's guide (Fig. 4).

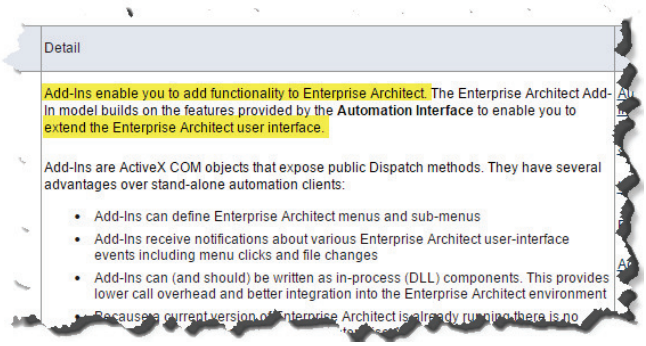

Fig. 4. Portion of User Guide (online) about "Add-in" functionality

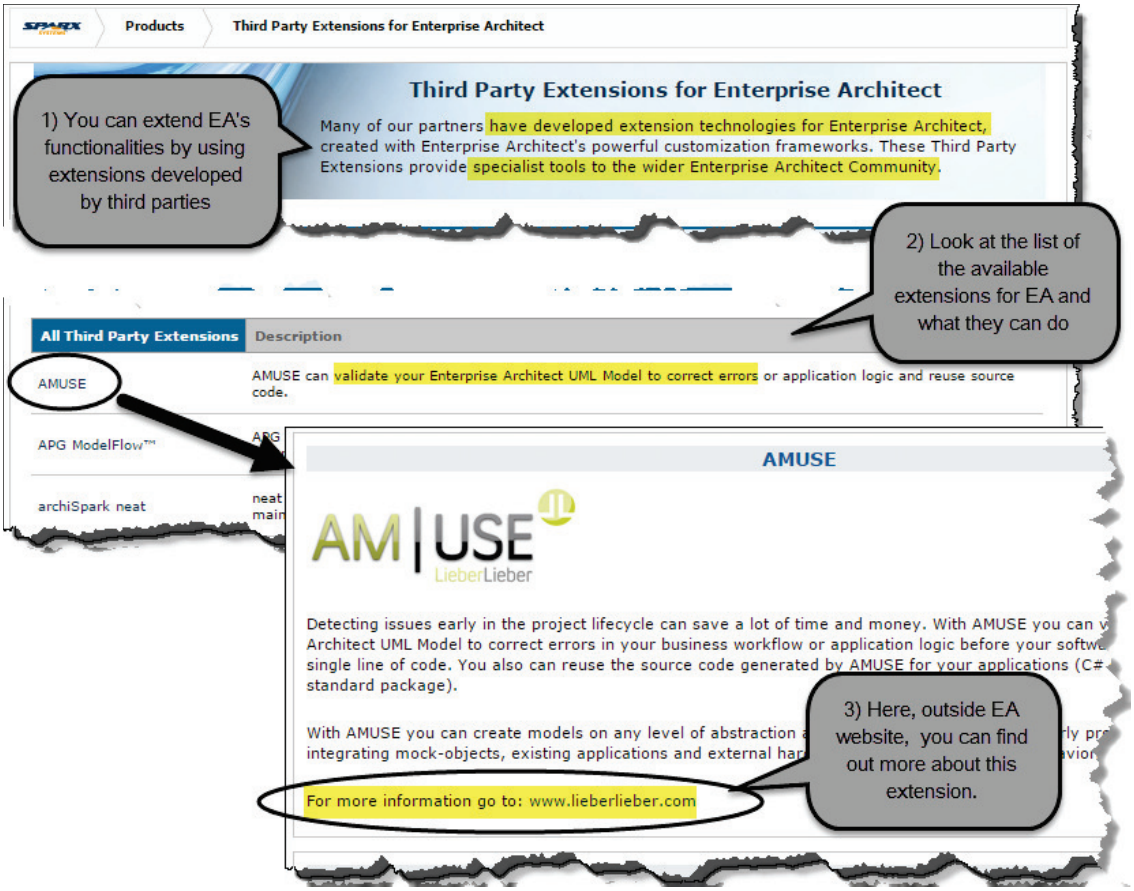

Fig. 5. Enterprise Architect Third Party Extensions – User Guide

The **metacommunication message** conveyed by metalinguistic signs was defined as follows:

(A) You are an experienced user of EA. You are also familiar with the use and creation of extensions and add-ins for tools that you may use, possibly a professional software developer. You know that extensions can be developed by third parties and that they typically provide documentation for their extensions on their websites.

(B) *You need to use or create extensions and add-ins to expand the functionalities or interface (elements and behaviors) provided by me to build activity models.*

(C) *You may want to use extensions created by me or third parties to expand the functionalities provided by me. And you may also want to create your own add-ins to expand the possible interface elements provided by me.*

(D) *I have therefore designed Extension and Add-In functionality to support you in modeling tasks. I indicate some Extensions developed by third parties. If you want to know about them, please visit my website. I give you some basic information about those extensions there and, if you want to know more I give you the link to the extension's owners' website. If you want to use other extensions, you should follow the pointers to where the information resides in the Internet, and then study the documentation in order to install the extension. I can help you with Add-ins. Please look for information in the User Guide.*

(E) *My vision is that you, just like me, are a professional software developer. As a professional developer, you need to be efficient, even as unpredicted situations arise. Therefore, I leave an open door in EA for your own extensions and additions. In this way you can customize EA and make it your own tool.*

Static signs analysis. Both extensions and add-ins are objects of metacommunication achieved with static signs (Fig. 6). A menu structure labeled "Extensions" contains two sub-lists of menu items separated by a horizontal line. This communicates that there are two different messages from the designer. The upper sub-lists is a mix of proper names and verbs, static signs that can be used to invoke (trigger) the corresponding functionality. Some of the signs in the list allow us to make reasonable sense of their meaning (*e. g.* "Publish"), whereas others are likely to be meaningless to the targeted user (*e. g. "CodeTrigger V4.2")*.

The lower sub list in Fig. 6 refers to add-ins. Unlike the upper sub-lists, this one contains two signs that are clearly related with each other ("Add-In Windows" and "Manage Add-Ins…").

When installed, EA already incorporates some extensions, which appear in the upper sub-lists of Fig. 6. If other extensions are incorporated to EA by the user, their names are inserted in the upper sub-lists. As a consequence, the static metacommunication does not distinguish between items recommended by the EA designer at installation time and items included by users at use time. Neither does it distinguish between items created by the user and items created by third parties nor even by the EA designer, as is the case of Extension menu items in the lower sub-lists shown in Fig. 6.

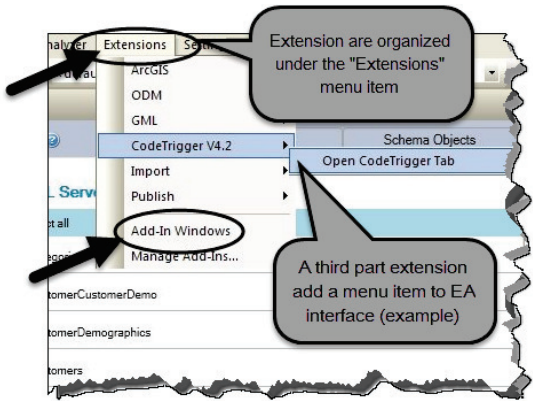

Fig. 6. Extensions organized in a specific menu item

The **metacommunication message** conveyed by static signs was defined as follows:

(A) You are familiar with the use and creation of extensions and add-ins for tools that you may use, possibly a professional software developer.

(B) *You need or want to use extensions and add-ins to expand the functionalities or interface (elements and behaviors) to build activity models. You are probably not intimidated by the complexity of using various extensions at the same time.*

(C) *You want to use the set of extensions that I provide for you. And you might also want to work with add-ins, but I don't think it is necessary to provide further information about them.*

(D) *I have therefore designed Extension and Add-In functionality to support you in modeling tasks. I offer you an initial list of Extensions (which can be expanded). I also indicate that you can work with add-ins.*

(E) *My vision is that you are more advanced than a novice user, so you need extensions and additions. Therefore I offer some extensions for you, but you need to get information about add-ins elsewhere.*

Dynamic signs analysis. In order to grasp the meaning of cryptic items in the Extensions menu the user may resort to dynamic signs (*i. e.* engage in interaction, select the menu item, interact with dialogs, see what happens and then try to make sense of what metacommunication tells him). One of the items ("Manage Add-Ins…") is followed by the standard indication of follow-up dialog ("…"), which communicates an invitation for the user to engage in interaction and unfold the meaning of an associated dynamic sign. However, after clicking on the menu item "Manager Add-ins…" the user does not get more information about it (Fig. 7). For the extensions, each one has a different sequence of screens presented once the user clicks the menu items. Some of those items open a screen, and the user needs to choose a file and inform specific settings to perform some task related to the extension (Fig. 8). The dynamic sign only presents the screen, considering that the user know what to do with it.

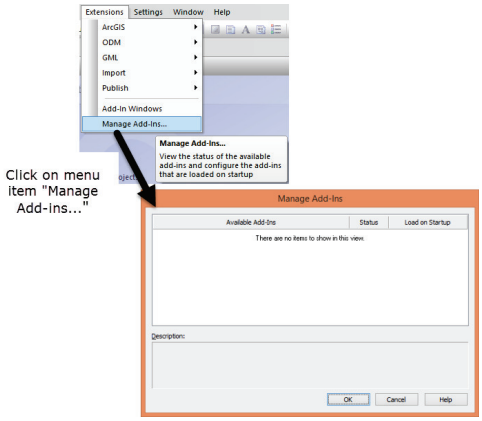

Fig. 7. Click on "Manage Add-ins..."

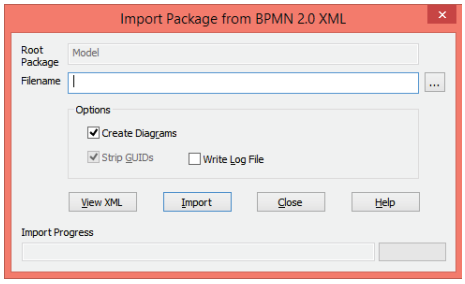

Fig. 8. Extension>Import> BPMN 2.0

The **metacommunication message conveyed by dynamic signs** was not completely filled out for the illustration case scenario ("*Problems with resources to extend/add functionalities*"). The dynamic signs did not communicate anything about portions **A, C, D** and **E** of the metacommunication template. The only portion filled out by dynamic signs is:

(B) *You might want or need extension and add-ins to build activity models.*

The message conveyed by dynamic signs is incomplete. This situation needs to be further investigated by the analyst in the next step. It might be related to a communication strategy of the EA's designer, who decided to convey his message using other classes of signs. Another possibility is that there might be a gap in the message, in the sense that the designer failed to communicate clearly to the user what his system does.

Issues detected in the ***segmented analysis*** (*e. g.* the presence of ambiguous, vague or incomprehensible signs) are included in a list for further evaluation with CDNf. All issues are accompanied by information about where supporting evidence for it has been found during the analysis.

C. *Step 2 – SIM Inspection: Contrast and Comparison*

In this step, the analyst takes the three segmented templates and, by contrast and comparison, he examines if messages conveyed by all templates are consistent. He also examines if communication contents appear in more than one template, which is an indication of redundancy. Finally, he verifies how, the content communicated in the templates is used to fill out the complete metacommunication template. This is a verification of how metacommunication content is distributed among the three classes of interactive signs.

Issues detected in this step (*e. g.* inconsistency, inadequate level of redundancy, lack of metalinguistic signs to explain static or dynamic interface signs) are included in the current list for further evaluation with CDNf. Again, all included issues are accompanied by information about where supporting evidence for it has been found during the analysis.

For the illustration case, EA's metacommunication is strongly associated to metalinguistic signs. There are static signs communicating about extensions and add-ins, but they communicate only parts of the message. The static signs need to be combined with metalinguistic signs for the designer's message to become entirely clear. The same happens with dynamic signs. The only portion of the message fully communicated by dynamic signs regarding extensions and add-ins is that the user wants or needs them while building an activity model (**B. *"…what I've learned you want or need to do…"***).

At the of this phase, in CSCE+TNP, we found communicative issues with EA's metacommunication about extensions and add-ins. Extensions are pieces of software developed by third parties, different from EA's designers, that can be plugged into EA. Add-ins are typically simpler functions that users themselves can create and incorporate to the original set of functions.

D. *Step 3 – SIM Inspection: Final Evaluation*

In this step, the analyst takes the results of previous steps to a higher level of abstraction. His aim is to evaluate if the designer's communicative strategies are good (*i. e.* likely to achieve the expected effects on the targeted user) or not, and why. The evaluation is organized around categories that emerge from the analyst's interpretation and reasoning. For example, at this stage the analyst may realize that the designer's communication is heavily dependent on dynamic signs. Compared with static signs, which might be able to achieve equivalent communication in a redesigned version of the interface, this decision implies that the user will often have to wait longer for communication to unfold before he can interpret what some particular interface message means. The analyst may then create a category of analysis named "interaction length", which he will now use to revisit prior findings and/or produce new ones.

Issues detected in this step (*e. g.* lack of informative and explanatory communication) are included in the current list for further evaluation with CDNf. Again, all included issues are accompanied by information about where supporting evidence for it has been found during the analysis.

For the illustration case, after SIM analysis, we have evidence indicating that the designer's communicative strategies were focus on another kind of users different than the user profile defined for the combined evaluation (novice user for EA). The user who is able to use extensions and create add-

ins are not novice for EA, he is an experienced user that knows EA enough to recognize when he needs other functionalities, different from those already provided by EA's designer.

At the end of the semiotic analysis with SIM, we also have a **list of communicability issues**, which will be the input for the cognitive analysis with CDNf. We identified three communicability issue:

Lack of differentiation for extensions' ownership. We learned that extensions may have been developed by EA designers and also by third parties, which are indicated in EA website. The default EA interface, which is the one presented to the user after installation, includes the extension provided by EA designers. However, if the user adds third parties extensions, there is no distinction between the two kinds of extensions. They are grouped under the same menu item and, by looking at the menu, the user cannot tell who is the owner of each extension. Over time, if many extensions are added, this situation can lead to confusion about the origin of extensions.

Extension's learning spread over different sources. We learned that the user needs to interact with different means to fully understand extensions. Sometimes the user needs just the resources in the EA interface, others he needs information from the EA website, and, in some cases, he may even need to visit third parties' website. This scenario imposes heavy learning costs for the user who wants to master EA extensions and add-in functionality.

User communicating with different designers. We also learned that the user might be communicating with different designers while interacting with EA. First, while interacting with functionality and extensions provided by the EA standard interface, the user is communicating with EA designers. Second, while interacting with third parties' extensions, the user is communicating with another design team. And, third, if he has or gains enough, expertise, he will be communicating with his own design, that is, 'talking to himself', so to speak. If he wishes, the user can create or change interface elements and behavior through way of add-ins. This is the mechanism that implements specific interfaces to extensions created by different authors. Therefore this issue is related to the first one, mentioned above. Note that third-party extensions do not necessarily follow the same interaction patterns. So, the user may have to face different signification systems and communicative strategies every time he switches from one extension to the other.

E. Step 4 – CDNf Inspection: Cognitive Dimensions Check

As shown in Fig. 1, the CDNf inspection carried out in CSCE is partial in two important ways. First, the first three steps from a standard CDNf analysis (Fig. 3) are skipped and the analyst uses the information available from the preparation phase, performed before the semiotic analysis, to define its ideal notation profile. Second, the object of CDNf analysis at this step is not the entire set of notational systems and information artifacts included in the software tool being analyzed, but only the portions of it where communicative issues have been detected.

Thus, for every item in the current list of issues, the analyst identifies the tasks and subtasks in it. Then, for every task or subtask at hand, given the profile of the targeted user and the set of notational systems with which the user must interact to carry them out, the analyst checks all fourteen cognitive dimensions (TABLE I) in search of evidence regarding the user's mental workload.

For the illustration case with EA, we identified cognitive *issues* that can be composed with the communicative issues identified at EA semiotic evaluation that are used as input to the CDNf inspection.

The problem of losing ownership distinctions of items included in the "Extension" ***("Lack of differentiation for extensions' ownership"*** issue) menu list is associated with CDN ***"Hidden Dependencies"***. By looking at the notation (a label), the user cannot tell who the owner of the extension is, that is, on whose decisions the meaning of the extension depends on. Of course, if the user himself has created the item, he may (or may not) remember that. Another cognitive dimension comes then into play, ***"Visibility"***, revealing the cognitive load imposed by the lack of a notational element that indicates which items have been created by the user, for example.

Another important contribution of CDNf for our analysis is to show the cognitive cost of "***Diffuseness***" of information if the user seeks to understand the meaning of the item "*CodeTrigger V4.2*", for example ("***Extension's understanding spread over different sources" issue***). The search begins in one notational environment (EA), then proceeds to a second one (the EA website), where a link refers the user to a third one (the extension owner's website). "***Diffuseness***" occurs when information is dispersed over an extended notational space.

Finally, converging with SIM results that showed how metacommunication coming from different designers ("***User communicating with different designers" issue***) can disrupt the structure of the EA designer's metacommunication message, CDNf confirms this fact with "***Consistency***". From a user's cognitive point of view, uncontrolled additions of notational systems into an existing notational system are a threat to consistency in the collection of information artifacts. By the same token, this lack of consistency can lead users into error, that is, we have to add the cognitive loads of yet another cognitive dimension, "***Error Proneness***".

F. Step 5 – CDNf Inspection: Comparison with Ideal Notation Profile

Once the analyst has scanned the entire list of communicative issues with the fourteen cognitive dimensions used in CDNf, he compares his findings with ideal situations, that is, notations that impose low cognitive workloads. He identifies all cognitive issues revealed in this comparison and adds them to the corresponding record of communicative issues detected by SIM in previous steps. If no cognitive issues are found, the record is not changed.

All issues detected in this step are added to the corresponding communicative issue record in the current list

for further evaluation with TNP. As before, all included issues are accompanied by information about where supporting evidence for it has been found during the analysis.

In EA's illustration case, at this step we could see which cognitive dimensions of notations were associated with the communicative issues reported by SIM. The dimensions did not point to further issues, but rather to further details about the notation used in the metacommunication message. Such is the case of "***Role Expressiveness***", for instance. This dimension refers to the ability of the notation to express *roles* of the object they refer. Our analysis showed that the "Extension" menu was in itself an information artifact (a unitary notational system) whose role is to congregate all unpredictable notations created at use time by means of including extensions or add-ins. Rather than a cognitive *issue*, this is a cognitive *feature*. Delimiting the space of metacommunication produced by *others* is a good strategy in design. However, in EA, this strategy needs improvement, as the communication issue

G. Step 6 - TNP Characterization: Check all three factors

In this step, we analyze the listed issue items against the "T", "N" and "P" parts of the triplet and the relations between them. This step effects the contextual binding of communicability issues, along with its associated cognitive extensions, in the modeling domain. We take into account the fact that people use tools and notations to build models that are, themselves, notational artifacts, and will be used by other people (or even the same people) at later stages of development process.

In EA's illustration case, the current list of issues with communicative (communicability) and cognitive (usability) annotations coming from SIM and CDNf inspections can be structured using all the three factors in TNP. Here is a portion of the instantiated structure:

T (EA) communicates Pd (the EA designer's) intention to provide Pu (EA users) with powerful resources to extend T (EA's) functionality at use time. Pu thus creates and includes new N (Extensions and Add-Ins) in T (EA) while manipulating T-N (EA's notational systems). However, some other Px (third-party extension owners) can interfere with Pd and Pu ongoing communication through N (EA's original interface). Px will use his own N (modeling language notations and interface controls), which will probably be different from the N (metacommunication message signs) with which Pd and Pu communicate, mediated by T (EA). If Pu needs clarification about new N (extensions and add-ins) included in T (EA), Pu will have to use other T (EA's website and Px's website), and their respective N.

H. Step 7 – TNP: Plotting findings onto the final metacommunication template

In the final step of the method, the analyst takes all findings and annotations coming from previous steps and tries to fill out a new version of the metacommunication template, considering the portions presented in TABLE II. Note that because the analyzed data comes from interactive breakdowns, alone, it is possible (and even desirable) that the template will have gaps in it. Gaps mean that no issues have been found regarding the corresponding content.

In EA's illustration case, in the last step of the analysis, the entire list of issues, structured with TNP elements and relations among them is plotted onto the metacommunication template. The template is a message reconstructed by the analyst, considering the list of HCI issues detected in EA's interface. The illustration of results is presented incrementally. The analyst now speaks for the designer, that is, he assumes the first person of discourse ("I", "my") and incorporates into the metacommunication message the acknowledgment of communicative and cognitive issues that the user (addressed as the second person in discourse, "you") will have to avoid, circumvent or resolve. In the case presented in this paper, we have breakdowns in all portions of the metacommunication template, as follows:

"Here is my understanding of who you are,". You are an experienced user of EA. You are familiar with the use and creation of extensions and add-ins for tools that you may use, possibly a professional software developer. You know that extensions can be developed by third parties and that they typically provide documentation for their extensions on their websites. You are probably not intimidated by the complexity of using various extensions at the same time. You can look for documentation to learn how to use each one of them on your own, and if errors occur you believe you can detect and correct them without my help.

"…what I've learned you want or need to do…" You want or need extensions and add-ins to expand the functionality or interface (elements and behaviors) provided by me to build activity models.

"… in which preferred ways, and why.". You may want to use extensions created by me or by third parties to expand the functionality I provide with EA. And you may also want to create your own add-ins to expand the possible interface elements I provide.

"This is the system that I have, therefore, designed for you, and this is the way you can or should use it…" I have therefore designed Extension and Add-In functionality to support you in modeling tasks. I already include in EA a couple of Extensions developed by me and indicate some developed by third parties. If you want to know about the last one, please visit my website. I give you some basic information about those extensions there and, if you want to know more, I give you the link to the extension's owners' website. If you want to use other extensions, you should follow the pointers to where the information resides on the Internet, then study the documentation to install the extension. Once it is installed, I give you easy access to it in the "Extensions" menu. All extensions installed are listed there. If you click on their names you will call the extension's interface, which you can use as instructed by owners. I cannot help you if errors occur during use. So, be sure to get yourself the necessary resources to use extensions effectively and efficiently. You can also create and manage Add-Ins with a simple tool I provide you. I assume that you will do this only occasionally, which means you can recognize – in the Extensions menu – which items are third-party's extensions and which ones are you're your own add-

ins. You can also manage your Add-ins with an Add-in Manager. I can help you with Add-ins. Please look for information in the User Guide.

"in order to fulfill a range of purposes that fall within this vision.". My vision is that you, just like me, are a professional software developer. Therefore, you are used to solving problems using your own resources rather than being helped by others. As a professional developer, you need to be efficient, even as unpredicted situations arise. Therefore, I leave an open door in EA for your own extensions and additions. In this way, you can customize EA and make it *your own tool*.

We can see by the partial template fill-out above that the effect of design choices is explicitly expressed in the metacommunication messages. The designer acknowledges his own values and beliefs in building EA, which may strongly contrast with the actual user's profile. One of the most obvious contrasts is the fact that in our inspection scenario, the user's profile was that of a *novice* EA user. Hence, the partial template above shows that the designer's entire message above is meant for *someone else*. Put in another way, if extensions and add-ins are fundamental for *any* modeler to use, then the conversation must be extensively redesigned to reach *novice* users. If not, one of the points that can be elaborated is the explicit communication of which conversations are for *novices* and which ones are for *experienced* users. If they are completely intertwined in the interface, then a *novice* may suddenly begin to talk about things that he or she cannot understand. In general, the organization of conversations *per profile* of the receiver is a major semiotic engineering task, where bad solutions typically charge their cost in terms of the user experience.

V. DISCUSSION

The combination of HCI inspections driven by communicability and usability achieved with CSCE+TNP produces rich characterizations of issues related to breakdowns identified during the interaction with modeling tools. CSCE incorporates the core steps of existing semiotic (SIM) and cognitive (CDNf) methods.

In view of our long-range goal in research, we focus on *issues* related to breakdowns occurring in interaction with modeling tools. This means that we do not produce a characterization of the entire metacommunication message, annotated with cognitive dimensions and structured with TNP relations. We work with problematic fragments of metacommunication. This is more efficient for the long-range goal of our research, but we cannot be sure of what we are missing because of our choice.

In CSCE+TNP execution, the semiotic analysis is carried out before the cognitive analysis. We decided for that order because SIM takes a much larger perspective on human-computer interaction (including the designer and his design rationale) than CDNf (which concentrates on users and notations). We must, however, acknowledge the possibility of the reversed order producing interesting results. SIM, in this reversed order, would impose an *a posteriori* intentional structure to a series of observations produced with CDNf. The ideal notations used in the final step of CDNf inspections might challenge the justification of communicative strategy choices, for example ("*if there is a better communication language, why have I not used it in my design?"*).

Both SIM and CDNf account for all factors of TNP, but from completely different perspectives. In the specific context of software modeling tools, the communicative perspective of SIM highlights "P" and "T" more than "N". In fact, SIM cannot produce the kind of detailed analysis that CDNf produces for notations. However, SIM can go into the details of various P-P relations, or P-T-P relations, which CDNf is not prepared to handle, as we showed with the illustration case. Moreover, the rich and contextualized collections of issues gathered with CSCE+TNP can be explored by software tools' designers for at least two purposes: (a) to solve the evidenced issues and; (b) to learn from those issues and think about the cause-effect relation of their design decisions and the user experience while building artifacts in the software development process.

Moreover, when the final metacommunication template is reconstructed by the analyst, in spite of its biases and gaps, the contribution of SIM and CDNf dimensions to the end result of the analysis becomes clearer. The reconstructed designer's message is now changed by concessions and justifications (or excuses) for materially supported evidence that some design choices cause problems for the users, whether within the limited scope of user-system interactions, or in the broader scope of interaction that includes its antecedent (the designer's decision) and consequent (the social purpose of using the technology).

As mentioned in the introduction section, the complete results achieved in the EA's study were divided into four breakdown categories and we only illustrated the results for one of them ***(Problems with resources to extend/add functionality).*** In TABLE III, presented at the beginning of the next page, we present the relation between breakdown categories (rows) and basic interaction elements accounted by semiotic and cognitive perspectives (columns). Items marked with "*****" have been added on each side by elements and relations in the *triplet*. Gray cells mark the presence of relevant findings in the study. "Signification System" refers to the nature of signs being used to convey the designer's message. So, for example, regarding the semiotic analysis, issues in this illustrated category were related to signification systems, 1st Person – DESIGNER and 2nd Person - USER at interaction time. To understand how the relation is established, think of the results illustrated in the previous section, especially as reported in the reconstructed metacommunication template. Regarding the cognitive analysis, this category was related to Interface Notation and 1st Person - USER (tasks/goals). Once again, the illustrated findings indicate how the relation is established. The TNP items were not directly related to that category. The illustration case does not bring in different software development team members, like evidence in other issue categories like "*Problems with supporting the software development process*". Hence, we cannot relate this category with the user's social role in the development process, for instance. The other rows in the table show how many relations have been established in the study and strengthen our argument that CSCE+TNP can produce a rich semiotic-cognitive,

psycho-social characterization of HCI design issues in the design of the inspected artifact's interface.

TABLE III – SEMIOTIC AND COGNITIVE REFERENCE FOR ANALYSIS

	Semiotic Perspective				**Cognitive Perspective**		
Issue categories/Analysis item reference	Signification System	1st Person - DESIGNER	2nd Person - USER at interaction time	***2nd Person - USER in a Social Context (Software Development Process)**	Interface Notation	***Modeling Notation (UML)**	1st Person - USER (tasks/goals)
Problems with resources to extend/add functionality							
Problems with resources for meaning construction with multiple notations							
Problems with resources for meaning construction with UML							
Problems with supporting the software development process							
* **Analysis item reference added by TNP *triplet***							

Since 2012, we have been developing and improving the combined semiotic-cognitive evaluation method for software engineering tools. The method has already been applied for evaluating visual programming notations [28], application programming interface languages [26], as well as other modeling tools [15][27]. We acknowledge that researchers with less experience in Semiotic Engineering may initially achieve only part of the results achieved by experienced researchers. However, because there is so much more to uncover and discover in this field of research, our priority at the moment is not to produce "off-the-self" solutions for software industry professionals to use right away. The value of our contribution, as we see it, is to advance research. Nevertheless, with the illustrations provided in the previous section, we believe that interested researchers can carry out studies of their own, especially – as is expected in research – if they resort to further publications about SIM [7][11][12] and CDNf [13][24][25].

VI. FINAL CONSIDERATIONS AND FUTURE WORK

In this paper, we have described and illustrated the procedures and results of the CSCE+TNP method. We have also discussed how some of its features (like focusing on breakdowns and working with a fixed semiotic-cognitive order of analysis) may impose limitations in the process of analysis. These are presumed limitations. There must be other limitations that we are aware of, however. One of the topics that, as we already know, deserve further studies is the fact – shared by most qualitative methods [31] – that our analysis is extensively dependent on *data* produced by the analyst. That is, in the process of interpretation, intermediate findings are taken as *input* (hence *data*) for subsequent stages of interpretation. The standard way to attenuate this effect in qualitative methods is triangulation. One way to triangulate CSCE+TNP findings is to contrast findings with those from studies with user observation methods, like the Communicability Evaluation Method [7]. We will then be able to see the validity of achieved results against stronger empirical evidence. Another form of triangulation is to contrast results achieved by multiple analysts. Our results have been critiqued by a peers (which is one of the forms to validate qualitative research [31]), but they haven't been contrasted with the results of analysis fully carried out by a second (or third) analyst. The contrast of independent results would certainly strengthen the conclusions we achieved. In sum, the most critical methodological limitations of our work have to do with knowing what kinds of triangulations are more productive to validate CSCE+TNP findings. Hence the importance of proceeding with our project and improve our knowledge. In this respect, we hope that other researchers will use CSCE+TNP, even if partially, in their investigations and provide us with an informed critique of the work we have done.

Regardless of limitations, however, CSCE in its current state can produce rich characterizations of HCI issues in the design of software modeling tools' interfaces. These characterizations are evidently better than the ones achieved with SIM or CDNf, individually. For this research, we focused on modeling tools, and that is why we used CSCE associated with TNP. However, we understand that CSCE can be used to evaluate other kinds of tools that support the software development process. Moreover, the TNP structure is a step in the direction of including the social dimensions of group work in the scope of analysis. It can, in fact, be viewed as a *light-*

weight version of group models that have been already used to investigate software development processes [32][33].

The limitations of this work open the path to further research and future work. We plan to investigate the method's potential to handle not only the fragments of faulty interaction evidence, but also the entire scope of user-system communication enabled by modeling tools in a particular scenario of inspection. This could be achieved by repeating the EA study with two evaluators, having the same technical capacity, one looking at the entire scope of interaction, while the other focuses on semiotic and cognitive issues. By comparing achieved results, we should know how much information about the design rationale we may be missing by not taking into consideration the positive aspects of interaction in the inspection scenario, since we only focused on issues, on negative aspects. Even if our long-term goal is unchanged, and we proceed to investigate the *Propagation Conjecture*.

Other interesting points to be investigated include the power of model-drawing tools, compared with semantic modeling tools. If the user has the freedom to add new notations or expand the provided notation, we might see the emergence of socially-negotiated secondary notations that can fill up expressive gaps of the base notation supported by the tool. This might give us an insight into the social protocols that are sometimes used to complement (and compensate for the limitations of) technological protocols commonly used in collaborative tasks such as software development [34].

Finally, we should not lose sight of why we are investigating the communicability and usability of modeling tools used in the software engineering process. Because models are in and of themselves a message about how an individual (or a group of) software designer(s) or developer(s) has (have) interpreted a certain aspect of the system that is under construction, all breakdowns encountered in modeling activities have the potential to impact the quality of the model, and thus the resulting system's end user experience once the development phase is concluded and the system is deployed. We actually do not know what this impact is or can be, but this is partly because we lack the appropriate instruments to track the problem from its origin. So, our research is an initial step in a long path that can lead us to know the answer for relevant HCI questions. In spite of all influences coming from our context of work – Software Engineering – we are investigating HCI topics, having to do with how, where, when and why interactive breakdowns can happen while software designers are acting as users of software development tools. Our next short-term steps involve new evaluations of modeling tools already inspected, but with different scenarios, to broaden our knowledge about the communicability and usability characteristics of those tools. Furthermore, we are planning to evaluate other modeling tools used during software development. We also intend to evaluate other kinds of tools that support software development process.

ACKNOWLEDGMENT

Juliana Ferreira thanks CAPES for a Ph.D. scholarship and CNPq for a postdoctoral scholarship supporting the continuity of this research. Clarisse de Souza thanks CNPq and FAPERJ for partially supporting her research on this topic. All authors thank the kind contribution of anonymous reviewers who helped them improve this paper. All remaining faults are, however, the authors' own responsibility.

REFERENCES

[1] D. Budgen, "The Cobbler's Children: Why Do Software Design Environments Not Support Design Practices?", Software Designers in Action: A Human-Centric Look at Design Work. Abingdon: Chapman and Hall/CRC, 2013, 199-215.

[2] R.S. Pressman, Software engineering: a practitioner's approach, 7th ed. NY: McGraw-Hill, 2010.

[3] G. Cockton, "Revisiting usability's three key principles." In Proceeding of CHI '08 Extended Abstracts on Human Factors in Computing Systems (CHI EA '08). ACM, New York, NY, USA, 2008, 2473-2484.

[4] J.D. Gould, Lewis, C. "Designing for usability: key principles and what designers think." Communications of the ACM, 28(3), 1985, 300-311.

[5] E.L. Hutchins, J.D. Hollan, D.A. Norman, "Direct manipulation interfaces", In D. A. Norman, S.W. Draper. User Centered System Design; New Perspectives on Human-Computer Interaction. L. Erlbaum Assoc. Inc., Hillsdale, NJ L. Erlbaum Assoc. Inc., Hillsdale, NJ , 1986, 87-124.

[6] B.E. John, D.E. Kieras, "Using GOMS for user interface design and evaluation: Which technique?", ACM Transactions on Computer-Human Interaction (TOCHI), v. 3, n. 4, 1996, p. 287-319.

[7] C.S. De Souza, C.F. Leitão, Semiotic engineering methods for scientific research in HCI. Princeton: NJ. Morgan & Claypool. 2009.

[8] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, R. Heldal, "Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?", MoDELS 2013: 2013, 1-17.

[9] OMG, Object Management Group -http://www.omg.org/

[10] OMG, Unified Modeling Language (UML), V2.4.1- http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF

[11] C.S., De Souza, C.F. Leitão, R.O. Prates, E.J. da Silva, "The semiotic inspection method", In Proceedings of 7th Brazilian Symposium on Human Factors in Computing Systems. New York, NY: ACM. ACM International Conference Series. pp. 148-157, 2006.

[12] C.S. De Souza, C.F. Leitão, R.O. Prates, S.A. Bim, E.J. Da Silva, "Can inspection methods generate valid new knowledge in HCI? The case of semiotic inspection." International Journal of Human-Computer Studies 68, 2010., 22–40.

[13] A.F. Blackwell, T.R. Green, "Notational systems–the cognitive dimensions of notations framework", HCI Models Theories and Frameworks Toward a Multidisciplinary Science, 2003, 103–134.

[14] Enterprise Architect Ultimate 11 - http://www.sparxsystems.com.au/products/ea/index.html

[15] J.J. Ferreira, C.S. De Souza, R.F.G. Cerqueira, "Characterizing the Tool-notation-people Triplet in Software Modeling Tasks", In Proceeding of the 13th Brazilian Symposium on Human Factors in Computing Systems (IHC'2014), Brazilian Computer Society, Foz do Iguaçu, Brazil, 2013, p. 31-40.

[16] J.J. Ferreira, Communication through models in the context of software development. Rio de Janeiro, 2015. 194p. D.Sc. Thesis - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 2015. (to appear in Portuguese)

[17] C.S. De Souza, The Semiotic Engineering of Human–Computer Interaction. Cambridge, MA. The MIT Press, 2005.

[18] M. Nadin, "Interface design: A semiotic paradigm." Semiotica 69.3-4, 1988, 269-302.

[19] M. Nadin, "Interface design and evaluation." In R. Hartson, D. Hix (Eds.) Advances in Human-Computer Interaction, vol. 2. Norwood, NJ. Ablex Publishing Corp. 1988, pp. 45-100.

[20] P.B. Andersen, A theory of computer semiotics. Cambridge University Press (2nd edition), Cambridge, 1997.

[21] C.S. Peirce, The essential Peirce: Selected Philosophical Writings. Vols. I , II. N. Houser and C. J. W. Kloesel (Eds.). Bloomington, IN. Indiana University Press, 1992-1998.

[22] U. Eco, The theory of signs and the role of the reader, The Bulleting of Midwest Modern Language Association. Vol. 14(1), 1981, pp. 35-45.

[23] J. Nielsen, "Usability inspection methods", In Proc. Conference companion on Human factors in computing systems, ACM, 1994, 413-414.

[24] A.F. Blackwell, C. Britton, A. Cox, T.R. Green, C. Gurr, G. Kadoda, R.M. Young, "Cognitive dimensions of notations: Design tools for cognitive technology", In: Cognitive Technology: Instruments of Mind. Springer Berlin Heidelberg, 2001. p. 325-341.

[25] T.R. Green, M. Petre, "Usability analysis of visual programming environments: a 'cognitive dimensions' framework", Journal of Visual Languages & Computing 7.2, 1996, 131-174.

[26] L.M. Afonso, R.F.G. Cerqueira, C.S. De Souza, "Evaluating application programming interfaces as communication artefacts", In Proceedings of PPIG'2012, London, UK, 2012, pp. 151-162.

[27] J.J. Ferreira, C.S. De Souza, "Communicating ideas in computer-supported modeling tasks: A case study with BPMN", In Human-Computer Interaction. Human-Centered Design Approaches, Methods, Tools, and Environments, Springer Berlin Heidelberg, 2013, 320-329.

[28] J.J. Ferreira, , C.S. De Souza, L.C.C. Salgado, C. Slavieiro, C.F. Leitão, F.F. Moreira, "Combining cognitive, semiotic and discourse analysis to explore the power of notations in visual programming", In Proceedings of VL-HCC'2012, 2012, 101-108.

[29] Gartner Group - https://www.gartner.com/doc/2601526

[30] J.M. Carroll, Making use: Scenario-based design of human–computer interactions. Cambridge, MA: MIT Press, 2000.

[31] J.W. Creswell, Research design: Qualitative, quantitative, and mixed methods approaches. Sage publications, 2013.

[32] R.M. Araujo, F.M. Santoro, P. Brézillon, M.R.S. Borges, M.G.P. Rosa, "Context Models for Managing Collaborative Software Development Knowledge", In: Modeling and Retrieval of Context (MRC2004), 2004, Ulm, Alemanha. CEUR Workshop Proceedings. Berlin: CEUR Workshop Proceedings, 2004. v. 114. p. 61-72.

[33] J. Grudin, "Groupware and social dynamics: Eight challenges for developers", Communications of the ACM, v. 37, n. 1, 1994. 92-105.

[34] C. Ellis, S. Gibbs, "Groupware: some issues and experiences", Communications of the ACM, 34, 1991.